The Thesis Committee for Dany M. Haddad

Certifies that this is the approved version of the following thesis:

Strengthening Weak Supervision for Information Retrieval

SUPERVISING COMMITTEE:

Joydeep Ghosh, Supervisor

Greg Durrett

Strengthening Weak Supervision for Information Retrieval

by

Dany M. Haddad

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2019

to myself.

Acknowledgments

This work would not have been possible without the support of Dr. Joydeep Ghosh. Dr. Ghosh, thanks for all the guidance, insightful chats, and especially, for all your great jokes. Looking forward to more to come. Discussions with the rest of the IDEAL group as well as my colleagues at CognitiveScale also helped shape the ideas presented here.

Thanks to Dr. Greg Durrett for the candid feedback and fruitful discussions which helped guide the project and inspired foundational portions of the proposed methods.

The contributions presented here build upon Dr. Hamed Zamani and Dr. Mostafa Dehghani's incredible work on weak supervision for neural IR. Hamed, thank you for your detailed responses, suggestions, and especially for encouraging me to submit my work for publication. Mostafa, thanks for being excited about my work from the beginning of the project and for your creative ideas.

And of course, the biggest thanks goes to my family and friends. My parents for their relentless support, encouragement and invaluable perspective. Thanks to The Robinson and the rest of the Home-Owner's Association for making Austin a great place to live. And finally, thanks to WNC$_{\text{ج}}$ and the larger WNCG with whose members many research (and non-research) ideas were discussed throughout the development of this work.

Strengthening Weak Supervision for Information Retrieval

Dany M. Haddad, M.S.E.

The University of Texas at Austin, 2019

Supervisor: Joydeep Ghosh

The limited availability of ground truth relevance labels has been a major impediment to the application of supervised machine learning techniques to ad-hoc document retrieval and ranking. As a result, unsupervised scoring methods, such as BM25 and TF-IDF, remain strong competitors to deep learning approaches whose counterparts have brought on dramatic improvements in other domains, such as computer vision and natural language processing. However, recent works have shown that it is possible to take advantage of the performance of unsupervised methods to generate training data necessary for learning-to-rank models. Surprisingly, machine learning models trained on this generated data can outperform the original unsupervised method. The key limitation to this line of work is the size of the training set required to surpass the performance of the original unsupervised method, which can be as large as 10^{13} training examples.

Building on these insights, this work proposes two methods to reduce the amount of training data required. The first method takes inspiration from crowdsourcing, and leverages multiple unsupervised rankers to generate soft, or noise-aware, training labels. The second identifies harmful, or mislabeled, training examples and removes them from the training set. We show that our methods allow us to surpass the performance of the unsupervised baseline with far fewer training examples than previous works.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Backround

Classical ad-hoc retrieval methods have relied primarily on unsupervised signals such as BM25, TF-IDF, and PageRank as inputs to Learning-To-Rank (LeToR) models. Supervision for these models is often supplied in the form of click-stream logs or hand-curated rankings, both of which come with their issues and limitations. First, both sources are typically limited in availability and are often proprietary company resources. Second, click-stream data is typically biased toward the first few elements in the ranking presented to the user [Ai et al., 2018] and are noisy in general. Finally, such logs are only available after the fact, leading to a cold start problem. These issues motivate the search for an alternate source of "ground truth" ranked lists for training LeToR models.

It has been shown that the output of an unsupervised document retrieval method can be used to train a supervised ranking model that outperforms the original unsupervised ranker [Dehghani et al., 2017c, Dehghani et al., 2017a, Zamani and Croft, 2017, Nie et al., 2018]. However, all these approaches share the requirement of training on an extremely large amount of data, in some cases as many as 10^{13} exam-

ples [Dehghani et al., 2017c].

This work develops methods that make more effective use of the generated training data, substantially reducing the number of training examples required. Two approaches are presented that make improvements in this direction, and beat the unsupervised method using fewer than 10% of the training rankings compared to that in prior art.

The first method proposed takes a crowdsourcing approach and collects the output of multiple unsupervised retrieval models, combining them in an optimal way. Following [Ratner et al., 2017], a joint distribution is learned over the outputs of said retrieval models and a new training set of soft labels is generated. The model created using this process is referred to as the *noise-aware* model. The *noise-aware* model does not require access to any gold labels. To differentiate them from labels originating from weak supervision sources, relevance scores assigned by a human are referred to as "gold" labels.

The second method builds on the idea of dataset debugging and identifies training examples with the most harmful influence [Koh and Liang, 2017] (the labels most likely to be incorrect) and drops them from the training set. In contrast to the experiments from [Koh and Liang, 2017], since the labels in the training set are known to be noisy, it is expected that a significant number of training examples will be mislabeled. The model learned using this approach is referred to as the *influence-aware* model.

Each of the proposed methods makes more effective use of the training data, reducing the computational burden of training the full scale model. Since the training set is known to be noisy, improving the quality of training labels as well as eliminating unconfident samples are two intuitively logical paths forward. This

work takes a theoretically principled, rather than heuristic, approach to achieving these objectives and presents experimental results supporting the efficacy of the proposed methods.

This thesis is structured as follows: first, the notation used throughout this work is presented followed by an overview of relevant background regarding neural Information Retrieval (IR) and weak supervision. The proposed methods are then described in detail followed by experimental results and evaluation.

1.0.1 Notation

Symbol	Meaning
y_i	The target value for the i^{th} datapoint.
x_i	Set of features for the i^{th} datapoint.
z_i	The i^{th} datapoint in either the training or test set. Typically a tuple containing the corresponding set of features, x_i and target y_i.
$\mathcal{Z}$	Dataset containing datapoints z_i.
$\mathcal{Q}$	A set of queries.
$\mathcal{D}$	A set of documents.
$\mathcal{X}$	The feature space of a retrieval model.
π	A ranking given as a permutation of items.
n	Number of training examples.
p	Number of parameters in the model under consideration.
m	Number of documents that appear in the final ranking.
θ	The set of parameters in the model under consideration.
$\mathcal{L}(\mathcal{Z}, \theta)$	Empirical risk function to be minimized with respect to the parameters θ.
$\hat{\theta}$	The final estimate of the model parameters $theta$ that minimizes the empirical risk $\mathcal{L}(\mathcal{Z}, \theta)$
rel_i	The relevance score of the item in the i^{th} position in the ranking under evaluation. In general $rel_i \in [0, 1]$
$[k]$	The set of integers from 1 to k.

.

1.1 Ad-Hoc Retrieval and IR background

Recent years have seen huge growth in the amount of available unstructured data, including text, audio, images and video [Croft et al., 2010]. Naturally, retrieving relevant content is an essential part of leveraging access to this data. Furthermore, as the amount of electronic data increases, the need for effective information retrieval methods becomes more dramatic. More specifically, the field of IR seeks methods of providing consumers with the content most relevant to their information need. This general goal can be reduced to more concrete tasks, such as question-answering and ad-hoc retrieval. This work primarily addresses ad-hoc document retrieval due to it's nature as a fundamental IR task.

Ad-hoc retrieval refers to the task of satisfying an information need specified by a text based search query. Generally, ad-hoc retrieval systems do not have access to context outside the query, and must meet the user's information need using only the text in the query itself and no features about the user. In contrast, other IR tasks might be able to leverage user-item interactions to take a collaborative filtering approach. In contrast to question-answering, ad-hoc retrieval is fundamentally a ranking problem and comes with it's own set of challenges.

1.1.1 Task Description

Ranking documents to satisfy a user's information need is a fundamental task for many IR problems. As the name of the task suggests, the query space in ad-hoc retrieval is unrestricted, and queries may be long or short and contain any number of uncommon terms. As a result, the distribution of queries has a very heavy tail. Accordingly, IR systems must be able to perform well on queries that appear rarely, or not at all, in query logs or historical data. As a result, many successful approaches

to the ad-hoc retrieval task leverage unsupervised techniques and heuristics that are robust to rare queries.

Vocabulary mismatch between the query and corpus of documents compounds the difficulty of dealing with rare query terms. More specifically, polysemy and ambiguity in language make the task of determining the relevance of a document to a query more challenging than simple exact phrase matching. In this light, unsupervised methods tend to fall short, often omitting relevant documents with small vocabulary overlap as well as overestimating the relevance of documents that are lexically similar, but semantically irrelevant.

IR Metrics

Since many IR tasks eventually involve generating a ranked list, retrieval results are usually assessed using ranking metrics rather than metrics such as Mean Squared Error (MSE) or recall which are more common in regression and classification tasks, respectively. While recall might still be relevant in some contexts, usually only the top-k ranked items are evaluated where k is much less than the total number of items to rank, or number of relevant items. As a result, one cannot hope to capture all relevant items in the top k in the case where there are more than k relevant items, so recall is no longer an informative metric. Precision remains valuable in a ranking context when a cutoff is k is considered. More specifically, when $rel_q(\pi, i)$ is the relevance score of the item in the i^{th} position of the ranking π to a query q, precision at a cutoff of k is evaluated as follows:

$$prec@k(q, \pi) = \frac{\sum_i^k 2^{rel_q(\pi,i)}}{k} \tag{1.1}$$

Intuitively, items that appear at the top of a ranked list are the most im-

portant, since a consumer will consider the items in ranked order. Accordingly, the quality of a ranked list can be assessed in a way that takes this into consideration. Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP) are two metrics that weight items near the top of the ranked list higher in their evaluation.

The NDCG of a ranking is evaluated as the Discounted Cumulative Gain (DCG) normalized by the best possible DCG achievable for query q at a cutoff of k. We refer to this optimal DCG as $IDCG@k(q)$.

$$\begin{aligned} DCG@k(q,\pi) &= \frac{\sum_i^k 2^{rel_q(\pi,i)} - 1}{\log_2(i+1)}, \\ IDCG@k(q) &= \max_\tau \frac{\sum_i^k 2^{rel_q(\tau,i)} - 1}{\log_2(i+1)}, \\ NDCG@k(q,\pi) &= \frac{DCG@k(q,\pi)}{IDCG@k(q)}. \end{aligned} \tag{1.2}$$

MAP is evaluated by taking the mean of the Average Precision (AP) over a set of queries. Given a set of queries $q \in \mathcal{Q}$ and corresponding rankings of n items $\pi_q \in \Pi$, MAP is evaluated as:

$$\begin{aligned} AP_q(\pi_q) &= \frac{\sum_i^n prec@i(q,\pi_q) \times rel_q(\pi_q,i)}{\sum_i^n rel_q(\pi_q,i)}, \\ MAP &= \frac{\sum_{q\in\mathcal{Q}} AP_q(\pi_q)}{|\mathcal{Q}|}. \end{aligned} \tag{1.3}$$

where $\sum_i^n rel_q(\pi,i)$ is the total number of relevant documents for that query.

1.1.2 Traditional Approaches

Despite their simplicity, Bag-of-words (BoW)[1] approaches to ad-hoc retrieval have enjoyed great success. Coupled with techniques such as stemming, they can effectively deal with the difficulty of retrieving relevant documents even in the context of rare queries. Methods such as topic-modeling, classical language-modeling techniques [Ponte and Croft, 1998] and Pseudo-relevance Feedback (PRF) [Lv and Zhai, 2009] attempt to overcome the vocabulary mismatch issue and have served as strong baselines for IR tasks. On the other hand, BoW methods often struggle to deal with the semantic meaning of a query, for example the queries "fish food" and "food fish" have the same BoW representation but correspond different information needs.

BM25, TF-IDF and Query Likelihood

Okapi BM25 [Robertson et al., 1995], TF-IDF and Query Likelihood (QL) [Ponte and Croft, 1998] are three strong baseline unsupervised ad-hoc retrieval models. Despite their simplicity, they have enjoyed much success as a fundamental component of approaches to many IR tasks. Generally, each of these methods scores a document's relevance to a query as a weighted sum of the terms that the two have in common. BM25 is generally viewed as a modified version of TF-IDF while QL takes a probablistic approach and models the document and query as arising from a generative process. The score of a document d to a query q, as given by these three retrieval models is presented below. The corpus of documents is given by $\mathcal{C}$ and documents and queries are each given by a collection of tokens.

[1] Techniques which treat text as an unordered collection of tokens

$$\begin{aligned} \text{idf}(t) &= \log\Big(\frac{|\mathcal{C}|}{\text{df}(t)}\Big), \\ score_{\text{tf-idf}}(d,q) &= \sum_{t\in q} \text{tf}(t,d) \times \text{idf}(t). \end{aligned} \tag{1.4}$$

where $\text{df}(t)$ is the number of unique documents that term t occurs in within the corpus $\mathcal{C}$ and $\text{tf}(t,d)$ is the number of times a term t appears in the document d. Variations on TF-IDF exist that incorporate ideas such as smoothing, to prevent the denominator from going to 0, and accounting for variations in document and query length. The score of a document for BM25 is given by:

$$\begin{aligned} \text{avgdl} &= \sum_{d\in\mathcal{C}} \frac{|d|}{|\mathcal{C}|}, \\ \text{idf}(t) &= \log\Big(\frac{|\mathcal{C}| - \text{df}(t) + 0.5}{\text{df}(t) + 0.5}\Big), \\ score_{BM25}(d,q) &= \sum_{t\in q} \text{idf}_{BM25}(t) \frac{\text{tf}(t,d) \times (k_1+1)}{\text{tf}(t,d) + k_1(1 - b + b \times |d|/\text{avgdl})}. \end{aligned} \tag{1.5}$$

Finally, the QL approach considers retrieval from a language modeling perspective and models the query as being generated by the same process that generated a document. Documents are ranked by the likelihood of the query under the language model learned for each of the documents. Assuming terms are generated independently, we have the following as the QL score:

$$p(q|M_d) = \prod_{t \in q} p(t|M_d) \prod_{t \notin q} (1 - p(t|M_d)),$$

$$score_{QL}(d, q) = p(q|M_d). \tag{1.6}$$

The document language model $p(\cdot|M_d)$ can be computed in a variety of ways, the maximimum likelihood estimate of a term t under the language model M_d is given by:

$$\hat{p}_{ML}(t|M_d) = \frac{\text{tf}(t, d)}{|d|}$$

As in [Metzler et al., 2004], the Indri retrieval model computes $p(t|M_d)$ as Dirichlet distributed. Selecting different values of α_t gives Laplace smoothing ($\alpha_t = 2$), Dirichlet smoothing ($\alpha_t = \mu \frac{\sum_{d \in \mathcal{C}} \text{tf}(t,d)}{\sum_{d \in \mathcal{C}} |d|} + 1$) or the maximum likelihood estimate ($\alpha_t = 1$):

$$p(t|M_d) = \frac{\text{tf}(t, d) + \alpha_t - 1}{|d| + \sum_{v \in \mathcal{V}} \alpha_v - |\mathcal{V}|}$$

Stopping and Stemming

In general, introducing further preprocessing steps can improve the performance of retrieval methods. Removing very common and uninformative words, referred to as *stopwords*, can help improve the performance of the QL retrieval model [Ponte and Croft, 1998], although for other unsupervised methods removing stopwords may have no significant impact [Croft et al., 2010].

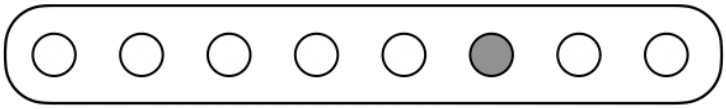

Figure 1.1: Vector representation of a word using one-hot encoding for a vocabulary that supports up to 8 unique tokens.

Since many retrieval methods rely on computing the overlap in vocabulary between the documents and the query, unifying tokens that contain the same root, or *stem*, can help improve performance. For example, removing suffixes such as 's', 'ed' or 'ing' from variations of the word 'call' can help improve a retrieval model's assessment of relevance.

Vector Space Models

As in other machine learning problems, a variety of vector representations have been proposed. From vector representations of words, representations for queries and documents can be constructed, for which many techniques have also been proposed [Le and Mikolov, 2014, Mitra and N Craswell, 2018]. The most straightforward representation for a word is to consider a fixed set of terms in a vocabulary, $\mathcal{V}$, and represent each word as a one-hot encoded vector $v \in \{0,1\}^{|\mathcal{V}|}$. Out of Vocabulary (OoV) terms are represented by a special "UNK" token. This one-hot vector can be used to construct a vector representation of a document by taking the sum of the vectors that it contains. In this representation, similar words, such as "cat" and "dog" are orthogonal; no notion of term similarity is encoded. Figure 1.1 shows a graphical depiction of the one-hot encoding of a word.

Inspired by the *distributional hypothesis* [Harris, 2015], Word2Vec [Mikolov et al., 2013] and GloVe [Pennington et al., 2015] each learn low dimensional word vectors which encode the distributional and semantic properties of words. It was shown empiri-

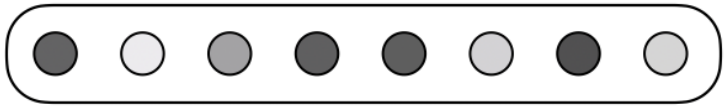

Figure 1.2: Dense vector representation of a word. Such representations can be learned with an algorithm like word2vec.

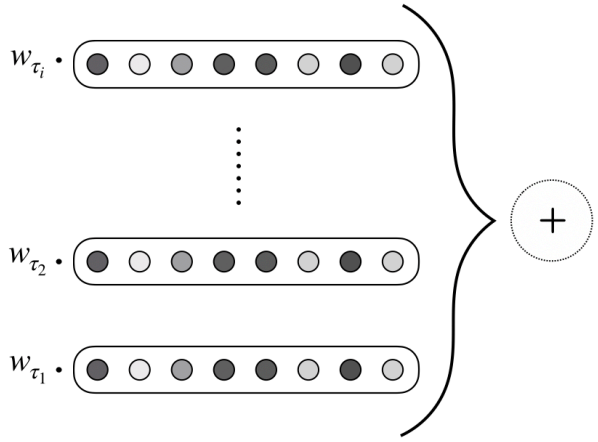

Figure 1.3: Combination of dense vector representation of words to form a vector representation of a document or query. Each token τ_i is optionally weighted by its importance, w_{τ_i}. Inverse document frequency is a common surrogate for token importance.

cally that in this space, vectors are compositional over addition, and that the inner product between pairs of words encodes a notion of similarity. These vectors can be learned using the target corpus, or some much larger source of text. In the IR context, these vector representations help levy the vocabulary mismatch between the query and documents to be ranked. Figure 1.2 shows a graphical depiction of a dense vector representation which can be combined as shown in figure 1.3 to form a vector representation for a query or document.

Other vector space models represent words using a collection of hand crafted features or latent representations learned from Latent Semantic Indexing (LSI) [Deerwester et al., 1990] or Latent Dirichlet Analysis (LDA) [Blei et al., 2002].

Learning-to-Rank

Supervised ranking approaches are canonically known as LeToR algorithms. LeToR requires a set of queries and documents along with corresponding ground truth relevance labels. The parameters of the LeToR model are learned by minimizing the loss over a set of queries and documents with known relevance labels. The form of this loss function categorizes the approach into either the *pointwise*, *pairwise*, or *listwise* paradigms.

In the pointwise approach, the loss function considers the score of an individual document in the ranking and is dependent on the score of that document rather than the score relative to other documents in the ranking. Furthermore, the pointwise approach does not consider the position of the documents in the ranked list. This approach tends to have poor generalization performance due to it allowing for an additional degree of freedom in the document scores; if all scores are rescaled, the ranking remains the same. Techniques such as monotone retargeting

[Acharyya et al., 2012] help avoid this issue.

Pairwise training compares the relative preference of one document over another in a given ranking. This eliminates the scaling issue that is inherent to the pointwise approach. The model reduces to a simple binary classification over pairs of documents, with $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n)$ training examples. Similar to the pointwise approach, the pairwise method does not incorporate query level information; more specifically, if the number of relevant documents for different queries varies widely, then the loss terms will be dominated by the queries with the most number of relevant documents. Query level normalization has been proposed to counter this effect [Cao et al., 2006].

Listwise approaches consider the entire ranked list at once and thus give context to each element being ranked. This allows the model to spend more capacity at higher positions in the ranking. More specifically, listwise methods directly optimize IR metrics such as NDCG or MAP. To tackle the non-differentiability of these metrics, approximate techniques such as SoftRank [Taylor et al., 2008] and LambdaRank [Burges, 2010] which is based on a heuristic that is shown to work well empirically. The trade-off is computational complexity during inference which is often $\mathcal{O}(m^2)$ or worse where m is the number of documents to rank.

1.2 Deep learning and Neural IR

In recent years, deep learning has led to dramatic improvements in Natural Language Processing (NLP) and computer vision tasks. However, the industry has yet to see the same level of improvement for IR problems. One likely explanation is the lack of sufficient ground truth data to train the model on. For example, the popular Robust04 TREC dataset contains only 250 queries with ground truth relevance

labels. With such a small dataset, it is difficult to train a model, let alone train and properly evaluate it's performance. In contrast to traditional LeToR approaches to IR, deep learning based methods learn document and query representations rather than working only with hand-crafted features. As a result, deep learning approaches have a much larger number of parameters and tend to require large quantities of training data compared to traditional techniques [Mitra and N Craswell, 2018].

While IR tasks bear some similarity to many NLP style problems, many of the challenges faced in the IR context are unique. For example, for ad-hoc retrieval, the retrieval and ranking system must be able to infer meaning and semantic relevance, deal with out-of-vocabulary tokens, and handle search queries of widely varying lengths. In addition, the ranking nature of IR tasks lends Machine Learning (ML) based approaches to different training procedures and considerations than models targeting NLP tasks. Furthermore, lexical matching has been shown to be a fundamental part of creating successful retrieval models [Croft et al., 2010]. As a result, deep learning architectures which leverage textual matching information between queries and documents tend to be common [Huang and Gao, 2013, Nie et al., 2018] unlike the sequence based models that are popular for NLP applications.

1.3 Weak Supervision for Retrieval

In order to sidestep the issue of limited ground truth data, Dehghani et al. use rankings generated by an unsupervised method as training data for a neural network ranking model [Dehghani et al., 2017c]. They build vector representations of queries and documents using a weighted sum of their constituent terms and pass the concatenated vector through a simple feed-forward network. Their model is

trained on approximately 10^{13} training examples and surpasses the performance of the original unsupervised method that generated the training data.

The model obtained from this method is referred to as the *rank* model for the duration of this work. The movement of data from documents and queries to training data for the neural network is shown in 1.4. For the *rank* model, the raw rankings from the unsupervised method (indicated by the Indri[2] Lemur logo in figure 1.4) are used directly to train the ML model. Chapter 2 introduces the methods which process these raw rankings before training (indicated by the '?' in figure 1.4).

Dehghani et al. explore pointwise, pairwise and listwise loss functions and show that the pointwise method struggles to generalize beyond the unsupervised ranker. In addition, they compare the performance of their proposed model to several baselines and show that it outperforms each of them significantly. The first baseline is a similar feed-forward network which instead uses hand crafted features. Another baseline they consider is a RankSVM [Joachims, 2002] model which uses the idf-weighted sum of word vectors as features. Since it is unclear how to update the parameters of the word vectors during the training of the RankSVM model they are pretrained on an external corpus and fixed as constant during training. Dehghani et al. conclude that the performance of their proposed method stems from:

- The dense, low dimensional query and document representations
- The ability of the feed-forward network to learn valuable transformations of the input features
- The choice of objective function

[2]https://www.lemurproject.org/indri.php

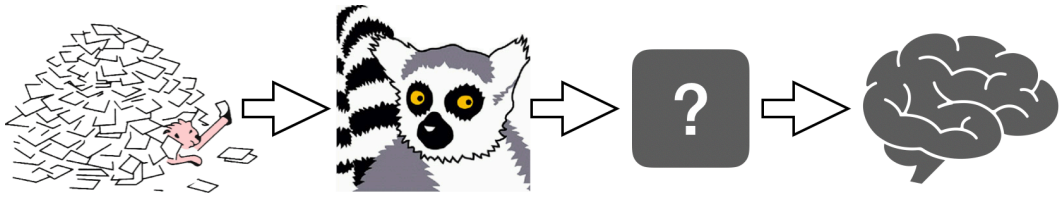

Figure 1.4: AOL Queries and TREC Discs 4-5 are passed into Indri for ranking using the QL retreival model. The output is transformed according to one of the methods described below and then used to train the network. The *rank* model uses the raw output from Indri as in [Dehghani et al., 2017c]. Based on NIST TREC logo and the Indri logo from the Lemur project.

- The size of the training set used

Zamani et al. learn vector representations of words for retrieval of relevant documents [Zamani and Croft, 2017]. Similar to how word vectors capture the distributional and statistical properties of words, relevance based word embeddings learn the relevance distribution [Lavrenko et al., 2001] associated with each word. These vectors are also trained using weak supervision (following [Dehghani et al., 2017c], we continue to refer to this as 'weak supervision' rather than 'distant supervision' since the difference between the two in the literature is blurred), using a similar dataset to [Dehghani et al., 2017c].

1.4 Snorkel

Consider the task of learning a classifier where the learner has access to multiple noisy labeling functions, or weak supervision source, each a mapping from the feature space to the set of classes. If these functions generate their outputs in a statistically independent way, the majority vote consensus gives the optimal combination in terms of accuracy [Berend and Kontorovich, 2013]. When labeling functions are correlated or exhibit some dependences, it may be possible to learn the dependence

structure between them and the combine the outputs of the individual weak supervision sources optimally.

Ratner et al. introduce a framework, which they name Snorkel, for combining multiple weak supervision sources into a single *noise-aware* set of labels [Ratner et al., 2017]. The fundamental idea is similar to ideas from crowdsourcing; the joint distribution over the outputs of these weak labelers is learned and then used to infer the true labels, which are treated as latent. In addition, Ratner et al. also learn the structure of the factor graph specifying the joint distribution without access to any ground truth labels by maximizing the marginal pseudo-likelihood.

[Ratner et al., 2017] investigates the impact of labeling density on the performance of Snorkel over taking the majority vote. The two extremes of the labeling density spectrum are the *high* and *low label density regimes*, which correspond to the case where every weak supervision source provides a label for each training point and when only a single source provides a label for each training point, respectively. They find that in the so-called *medium label density regime*, the middle ground between these two cases, the benefit of using noise aware labels is most pronounced.

1.4.1 Training a Discriminative Model

Formally, Snorkel aims to evaluate $p(Y = y|\Lambda)\forall y \in \mathcal{Y}$ where $\mathcal{Y}$ is the set of feasible values for y and $\Lambda \in \mathcal{Y}^k$ is the set of labels given by each of the k individual weak supervision sources. Once $p(Y|\Lambda)$ is known, it can be used to generate *noise-aware* labels. The loss minimized by Snorkel, with respect to some parameter w, in training the final discriminative model is then given by:

$$\ell_{\text{noise-aware}}(x, \Lambda; w) = \mathbb{E}_{Y|\Lambda}\ell(x, Y; w) \tag{1.7}$$

where $\ell(\cdot, \cdot; w)$ is the original loss function eg. cross-entropy.

1.4.2 Learning the parameters of the factor graph

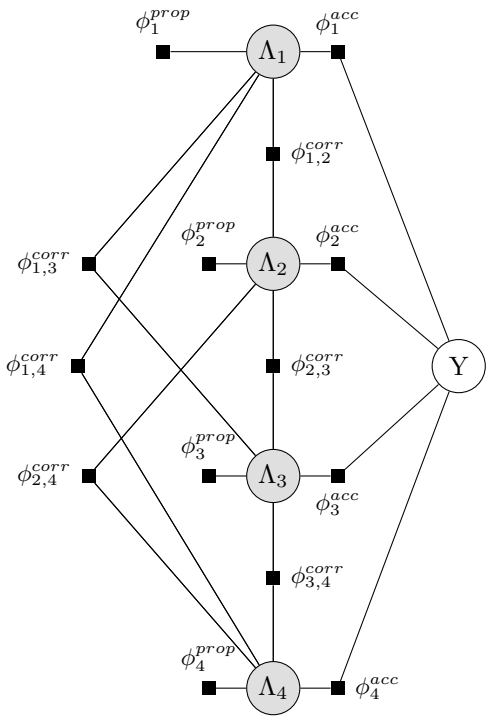

Figure 1.5: A typical factor graph showing the dependence of the labeling functions for four labelers which Snorkel learns a joint distribution over. The grey nodes are values for which we have a sample set. The white nodes are latent variables. The black squares are factors that are functions of the nodes they share an edge with. The joint distribution is given by the normalized product of these factors as in equation 1.8.

In order to compute $P(Y|\Lambda)$, the joint distribution is modeled as a factor graph, an example of which is shown in figure 1.5, whose factors are parameterized by w. The individual factor functions are given by ϕ. In general, for a single

datapoint, the joint distribution of Y, Λ parameterized by w is given by:

$$P_w(Y, \Lambda) = \frac{1}{Z(w)} \exp\left(w^T \phi(\Lambda, Y)\right) \tag{1.8}$$

where $Z(w)$ is the partition function given by:

$$Z(w) = \sum_{\Lambda} \sum_{Y} \exp\left(w^T \phi(\Lambda, Y)\right)$$

Since the true label Y is unknown, it is treated as latent when learning w. Marginalizing out Y in the likelihood function gives an objective that can be optimized with respect to w:

$$\mathcal{L}_{\text{snorkel}}(\Lambda; w) = \log \sum_{Y} P_w(\Lambda, Y)$$

This marginal likelihood is maximized with respect to w using stochastic gradient descent. Computing the gradient with respect to w gives:

$$\nabla_w \log \sum_{Y} P_w(\Lambda, Y) = \nabla_w (\log \sum_{Y} w^T \phi(\Lambda, Y) - \log Z(w)), \tag{1.9}$$

$$= \mathbb{E}_{Y \sim P_w(\cdot|\Lambda)}[\phi(\Lambda, Y)] - \mathbb{E}_{\Lambda, Y \sim P_w}[\phi(\Lambda, Y)]. \tag{1.10}$$

The expectations may be intractable for a large number of labeling functions and large target space. They may be computed via Gibbs sampling by iterating through the nodes in the factor graph and sampling from the conditional distributions; after burn-in, samples will be drawn from the joint distribution. For a small number of labelers and binary Y, this expectation can be computed directly.

In general, maximizing the marginal likelihood may not provide parameters that generalize to other datasets. In this case however, it makes sense to maximize

the marginal likelihood since the individual labelers are assumed to be predictive of the true y labels.

1.5 Influence Functions

As part of understanding model behavior, a practitioner may seek to quantify the impact that a specific training point has on the final learned parameters. The naïve approach is to retrain the model with every possible subset of training data and evaluate the model's performance on the target metric. Obviously, this is infeasible, in general. A similar approach would be to instead make the assumption that the impact of a training point does not depend on the rest of the training set. Although this is not a valid assumption in general, intuitively, it makes sense that the most impactful training points, such as outliers and mislabeled training points, will remain impactful regardless of the distribution of the rest of the dataset. Given n training points, this leave-one-out training approach yields n evaluations of the target metric, allowing the training points to be ranked by their approximate impact as given by the change in the target metric when excluding that training point; see equation 1.11. Even this simpler approach is still infeasible, especially in the context of large datasets and complex models which have long training times.

$$\mathcal{I}_{\text{leave-one-out}}(z_i; \mathcal{Z}_{test}) = \eta(f_{\setminus z_i}, \mathcal{Z}_{test}) - \eta(f, \mathcal{Z}_{test}) \tag{1.11}$$

Where f is the resultant model trained on the entire dataset, $f_{\setminus z_i}$ is the same model except trained on the subset of the training set lacking training point $z_i = (x_i, y_i)$ containing features x_i and target y_i. The test set is notated as $\mathcal{Z}_{test}$ and the target metric of a model f evaluated over a dataset $\mathcal{Z}$ is written as $\eta(f, \mathcal{Z})$.

A typical use case would select η as equivalent to the loss.

In the context of generalized linear models, statistical leverage and Cook's distance [Cook and Weisberg, 1982] from classical statistics describe how much impact a specific training point has on the learned parameters of the model. [Cook, 1986] provides an expression for the change in the trained model's loss at a test point due to an infinitesimal change in weighting of a specific training point in the empirical risk objective function. Using the loss at a test set as the target metric, Koh et al. [Koh and Liang, 2017] apply this idea to dataset debugging and prioritize the inspection of training points based on their influence on the model's performance. This approximates the process involving leave-one-out retraining described above. [Khanna et al., 2018] take a data summarization approach using Fisher kernels to embed data points in the space induced by the trained model; this approach turns out to be a generalization of the influence functions methods of Koh et al. These works show that evaluating the influence of each training point is an accurate surrogate for its impact on the learned parameters and can be uesd to help identify mislabeled training points. Retraining the model with the appropriate corrections made improves the performance of the model.

Starting from the influence of upweighting a single training point z_{train} by ϵ on the learned parameters [Cook, 1986] derives the following expression for the change in the learned parameters due to an infinitesimal upweighting of the loss at z_{train}:

$$\frac{d\hat{\theta}(\epsilon, z_{train})}{d\epsilon}|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_{\theta}L(z_{train};\hat{\theta}) \tag{1.12}$$

where $\hat{\theta}(\epsilon, z_{train})$ is the minimizer of the weighted empirical risk given by:

$$\hat{\theta}(\epsilon, z_{train}) = \text{argmin}_\theta \ (1+\epsilon)L(z_{train};\theta) + \frac{1}{n-1}\sum_{i}^{n-1} L(z_i;\theta)$$

and the Hessian of the empirical risk at $\hat{\theta}$ is given by:

$$H_{\hat{\theta}} = \frac{1}{n}\sum_{i}^{n} \nabla^2 L(z_i;\hat{\theta}) \tag{1.13}$$

Note that 1.12 is exactly the steepest descent direction at z_{train} in the quadratic norm the defined by $H_{\hat{\theta}}$ [Boyd and Vandenberghe, 2004]. Using the chain rule, Koh et al. then derive the change in the loss at a test point z_{test} due to an infinitesimal upweighting of the loss at z_{train}:

$$I_{loss}(z_{test}, z_{train}) = -\nabla_\theta L(z_{test};\hat{\theta}) H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{train};\hat{\theta})$$

Making a linear approximation to the change in the loss due to dropping a training point entirely from the training set gives:

$$I_{drop}(z_{test}, z_{train}) = \frac{1}{n}\nabla_\theta L(z_{test};\hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{train};\hat{\theta}) \tag{1.14}$$

Since $\frac{1}{n}$ is the weight of z_{train} in the original objective function, reweighting it by $-\frac{1}{n}$ cancels out the contribution from that training example.

For models with many parameters, computing I_{drop} directly for each training point is expensive due to the difficulty of computing and storing the Hessian H_θ (which contains $\mathcal{O}(p^2)$ entries where p is the number of parameters in the model) and it's inverse. To overcome this, [Koh and Liang, 2017] propose to estimate the inverse Hessian vector product in 1.14 directly. They propose two methods to achieve this,

one using a conjugate gradient solver, and a stochastic estimation method which recursively computes the Hessian vector product (HVP).

Koh et al. show several applications of their proposed technique, including identifying mislabeled training examples, model interpretability and even creating training set poisoning attacks.

1.6 Related Work

Much of the prior work in handling noisy datasets has been in the context of learning a classifier from noisy labels. In the binary classification context, noise is seen as a class-conditional probability that an observed label is the opposite of the true label [Jiang et al., 2017, Northcutt et al., 2017]. This *flip-probablity* is often estimated by cross validation, assuming the practitioner has access to a clean dataset with trusted labels.

Binary classifiers which output class probabilities, automatically induce a ranking on the set of elements being classified, and thus act as pointwise rankers. However, in the ranking context, one typically expects that models trained using pairwise or listwise loss functions will far outperform pointwise approaches [Liu, 2009]. As the names suggest, pairwise approaches consider pairs of candidates while listwise approaches consider the complete ranking. Since the label of a pair is determined by the ordering of the documents within the pair, it is not immediately obvious how the class-conditional flip probabilities from [Jiang et al., 2017, Northcutt et al., 2017] translate to this formulation. The relationship to listwise objectives is not straightforward either.

In [Dehghani et al., 2017a] and [Dehghani et al., 2017b], the authors introduce two semi-supervised student-teacher models where the teacher-model weights

the contribution of each sample in the student-model's training loss based on its confidence in the quality of the label. They train the teacher on a small subset of gold labels and use the model's output as confidence weights for the student-model. [Dehghani et al., 2017a] shows that using this approach, they can beat the unsupervised ranker using $\sim$ 75% of the data required when training directly on the noisy data. They train a cluster of 50 gaussian processes to form the teacher annotations which are used to generate soft labels to fine-tune the student-model. [Dehghani et al., 2017b] trains a so called *confidence network* which plays the role of the teacher-model providing the annotations. Similarly, the confidence network is also trained on a small set of gold labels.

Similar to approaches from crowd-sourcing, Ratner et al. transform a set of weak supervision sources, that may disagree with each other, to create a unified set of higher quality soft labels for the target task [Ratner et al., 2017]. This is achieved by learning a generative model for the noisy annotation process and aggregating the weak supervision sources into a single dataset. This new set of soft labels is then used to train a discriminative model. Their experimental results show that this approach outperforms the naïve majority voting strategy for generating the target labels. This inspires the *noise-aware* approach introduced in this work that involves modeling the joint distribution over a set of unsupervised rankers.

Koh et al. apply classical results from regression analysis to approximate the change in loss at a test point caused by removing a specific point from the training set [Koh and Liang, 2017]. They show experimentally that, even for highly nonlinear models, such as GoogLeNet, their method can approximate this change in loss well. They also apply their method to prioritize training examples to check for labeling errors. The *influence-aware* approach introduced in this work uses

influence functions [Koh and Liang, 2017] to identify and drop training examples with the largest harmful influence. Intuitively, these harmful training examples are expected to be mislabeled.

Chapter 2

Proposed Methods

This section discusses the two proposed methods for improving upon the *rank* model of Dehghani et al. More specifically, each method reduces the amount of noisy training data required to train the LeToR pairwise model introduced in [Dehghani et al., 2017c]. This work focuses primarily on pairwise formulation (shown in figure 2.2) since they typically lead to better performing models than the pointwise approach (shown in 2.1). Listwise approaches, although typically the most effective, tend to have high training and inference time computational complexity due to their inherently permutation based formulations [Liu, 2009].

2.1 Model Architecture

The baseline model considered is a slight variant of the *Rank* model proposed in [Dehghani et al., 2017c]. The tokens in the i^{th} query are represented as t_i^q and the tokens in the i^{th} document as t_i^d. Tokens are embedded in a low dimensional space using the mapping $E : \mathcal{V} \mapsto \mathbb{R}^l$ where $\mathcal{V}$ is the vocabulary and l is the embedding dimension. As in [Dehghani et al., 2017c], token dependent weights $W : \mathcal{V} \mapsto \mathbb{R}$ are

also learned to provide more flexibility to the BoW model. The final representation for a query q is a weighted sum of the word embeddings:

$$v_q = \sum_{t \in t^q} \tilde{W}_q(t) E(t) \tag{2.1}$$

where $\tilde{W}_q$ indicates that the weights are normalized to sum to 1 across tokens in the query q using a softmax operation:

$$\tilde{W}_q(\cdot) = \frac{\exp(W(\cdot))}{\sum_{t \in t^q} \exp(W(t))} \tag{2.2}$$

The vector representation for documents is defined similarly:

$$v_d = \sum_{t \in t^d} \tilde{W}_d(t) E(t) \tag{2.3}$$

In addition, the elementwise difference and products of the document and query vectors are concatenated into a single vector $v_{q,d} = [v_q, v_d, v_q - v_d, v_q \odot v_d]$ which is used as the final representation. The relevance score of a document, d, to a query, q is computed by passing $v_{q,d}$ through a feed-forward network with $ReLU$ activations and scalar output. As in [Dehghani et al., 2017c], a $tanh$ at the output of the *rank* model is used to coerce the values into the range $\{-1, 1\}$. For the other models discussed in this section the raw logit scores are used directly. The output of the model under consideration parameterized by θ is written as $f(x; \theta)$.

The training set denoted by $\mathcal{Z}$ is a set of tuples $z = (q, d_1, d_2, s_{q,d_1}, s_{q,d_2})$ where s_{q,d_i} is the relevance score of d_i to q given by the unsupervised ranker. The pairwise objective function minimized is given by:

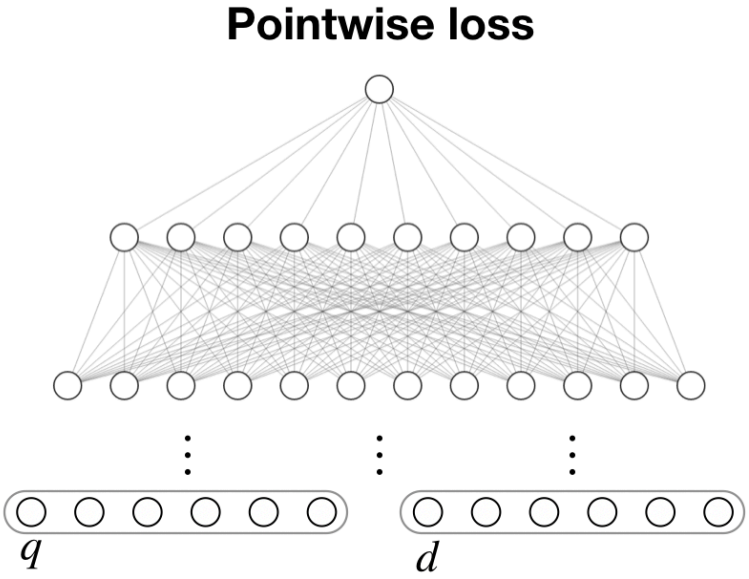

Figure 2.1: Architecture of the pointwise formulation. The score of a document is predicted given the vector representation of the document and query.

$$\mathcal{L}(\mathcal{Z};\theta) = \sum_{z \in \mathcal{Z}} L(f(v_{q,d_1};\theta) - f(v_{q,d_2};\theta), rel_{q,(d_1,d_2)}), \tag{2.4}$$

$$L_{ce}(x,y) = y \cdot \log(\sigma(x)) + (1-y) \cdot \log(1-\sigma(x)), \tag{2.5}$$

$$L_{hinge}(x,y) = \max\{0, \epsilon - \text{sign}(y) \cdot x\}. \tag{2.6}$$

Where $rel_{q,(d_1,d_2)} \in [0,1]$ gives the target relative relevance of d_1 and d_2 to q. L is either L_{ce} or L_{hinge} for cross-entropy or hinge loss, respectively. As in [Dehghani et al., 2017c], the *rank* model is trained by minimizing the max-margin loss and $rel_{q,(d_1,d_2)}$ is computed as $\text{sign}(s_{q,d_1} - s_{q,d_2})$.

Despite the results in [Zamani and Croft, 2018] showing that the max-margin loss exhibits stronger empirical risk guarantees for ranking tasks using noisy training

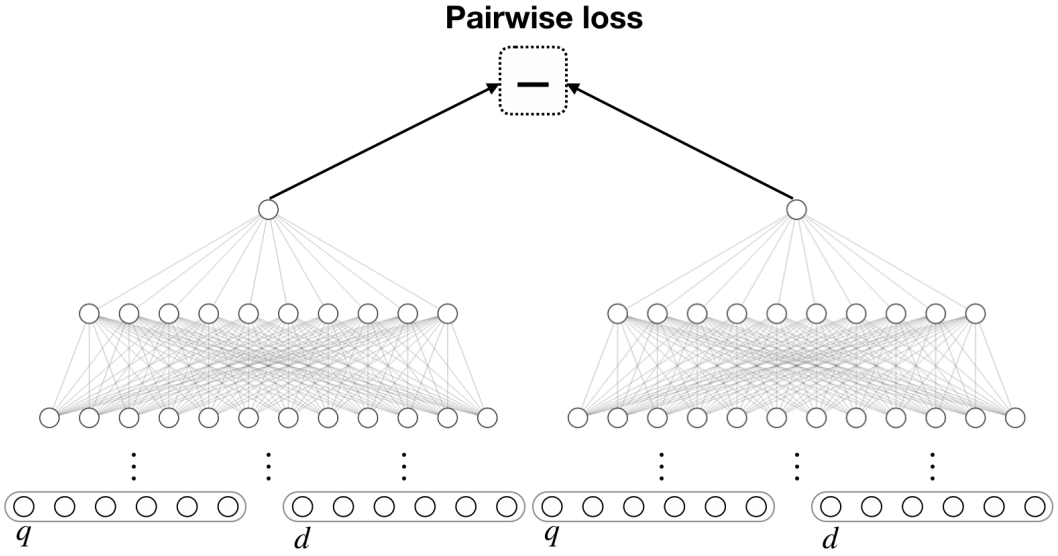

Figure 2.2: Architecture of the pairwise formulation. The same set of parameters are used for both feed-forward networks. The output is a scalar which is used to compute the relative preference of one document over another for a given query.

data, the cross-entropy loss is minimized in each of the models proposed in this work for the following reasons: in the case of the *noise-aware* model, each of the soft training labels are a distribution over $\{0, 1\}$, so a calibrated model is desired rather than one which maximizes the margin (as would be achieved using a hinge loss objective). For the *influence-aware* model, the cross-entropy rather than the hinge loss is minimized since the method of influence functions relies on having a twice differentiable objective.

2.2 Noise-aware model

In the *noise-aware* approach, $rel_{q,(d_i,d_j)} \in [0, 1]$ are soft relevance labels. For each of the queries in the training set, the top documents are ranked by relevance using k unsupervised rankers. Considering ordered pairs of these documents, each ranker

			Λ			
Query	d_1	d_2	**QL**	**QL+RM3**	**BM25**	**TF-IDF**
"post office"	DOC-1	DOC-201	1	0	1	-1
	DOC-1	DOC-10	0	1	1	1
	DOC-1	DOC-21	-1	-1	-1	-1
	DOC-49	DOC-32	0	0	0	1
		⋮		⋮		
"easter day"	DOC-48	DOC-322	-1	0	1	1
	DOC-48	DOC-395	1	1	-1	1
		⋮		⋮		

Figure 2.3: An example section of the Λ matrix created for the *noise-aware* model.

(more generally in the Snorkel context referred to as a labeling function) gives a value of 1 if it agrees with the ordering, -1 if it disagrees and 0 if neither document appears in the top 10 positions of the ranking. Specifically, each of the k rankers is a labeling function defined as a mapping from the feature space to the set of binary preferences in addition to an abstention class (indicated by 0): $\lambda_j : \mathcal{X} \mapsto \{-1, 0, 1\}$. The outputs of these labeling functions over the set of ordered pairs of documents are collected into a matrix $\Lambda \in \{-1, 0, 1\}^{m \times k}$ for m document pairs, shown in figure 2.3. The joint distribution over these pairwise preferences and the true pairwise orderings y is given by:

$$P_w(\Lambda, y) = \frac{1}{Z(w)} \exp(\sum_{i}^{m} w^T \phi(\Lambda_i, y_i)) \tag{2.7}$$

Where w is a vector of learned parameters and $Z(w)$ is the partition function. A natural choice for ϕ is to model the accuracy of each individual ranker in addition to the pairwise correlations between each of the rankers, see figure 1.5. So for the i^{th} document pair, we have the following expression for $\phi_i \coloneqq \phi(\Lambda_i, y_i)$:

$$\phi_i = [\{\Lambda_{ij} = y_i\}_{1 \leq j \leq k} || \{\Lambda_{ij} = \Lambda_{il} \neq 0\}_{j \neq l}]$$

Since the true relevance preferences are unknown, they are treated as latent. The parameters for this model are learned without any gold relevance labels y by maximizing the marginal likelihood (as in [Ratner et al., 2017]) given by:

$$\max_{w} \ \log \sum_{y} P_w(\Lambda, y) \tag{2.8}$$

We use the Snorkel library[1] to optimize equation 2.8 by stochastic gradient descent. Once the parameters of the model have been determined, the posterior probabilities $P_w(y_i|\Lambda_i)$ can be evaluated and used as soft training labels:

$$P_w(y_i|\Lambda_i) = \frac{P_w(y_i, \Lambda_i)}{\sum_y P_w(y, \Lambda_i)} \tag{2.9}$$

The same model architecture as the *rank* model is used. The algorithm for training the *noise-aware* model is given in algorithm 1.

Algorithm 1: Training the *noise-aware* model

Input: k noisy labeling functions $\lambda_j : \mathcal{X} \mapsto \{0, 1\}$;
set of document pairs $\mathcal{D}_{train}$ for training the discriminative model;
matrix Λ indicating ranker preferences for each document pair used for training the generative model
Output: The trained *noise-aware* model

```
Learn the weights, w, of the factor graph by optimizing (2.8)
Z_noise-aware ← {} for z_train_i ∈ Z_train do
    rel ← P_w(1|{λ_j(x_train_i)}_{j∈[k]}) as in 2.9
    Z_noise-aware ← Z_noise-aware ∪ {(x_train_i, rel)}
end
Train the model on the dataset with the noise-aware labels, Z_noise-aware
```

[1] https://github.com/HazyResearch/snorkel

2.3 Influence-aware Model

The *influence-aware* approach uses the methods of [Koh and Liang, 2017] to identify training examples that hurt the generalization performance of the trained model. Since the labels in the training set are noisy, it is expected that the training examples with the most hurtful influence are in fact incorrectly labeled. Accordingly, the model will perform better if we drop these incorrectly labeled examples from the training set.

The influence of removing a training example z_i on the trained model's loss at a test point z_{test} is computed as $\mathcal{I}_{drop}(z_{test}, z_{train})$ according to (1.14). Summing this value over the entire test set gives us $\mathcal{I}_{drop}(z_i)$. We compute $\mathcal{I}_{drop}(z_i)$ for each training example z_i, expecting it to represent z_i's impact on the model's performance at test time. In our setup, we know that some of our training examples are mislabeled; we expect that these points will have a large negative value for $\mathcal{I}_{drop}(z_i)$. Of course, for a fair evaluation, the z_{test} points are taken from the development set used for hyperparameter tuning (see section 3).

The computational difficulties of computing (1.13) are levied by treating the trained model as a logistic regression on the bottleneck features of the feed-forward network. All model parameters except the last layer of the network are frozen, and the gradient is computed with respect to these parameters only, as shown in figure 2.4. This gradient can be computed in closed form in an easily parallelizable way on modern GPU architectures, making techniques that rely on autodifferentiation operations [Pearlmutter, 1994] unnecessary.

$$s_{test} = H_\theta^{-1} \nabla_\theta L(z_{test}; \hat{\theta}) \tag{2.10}$$

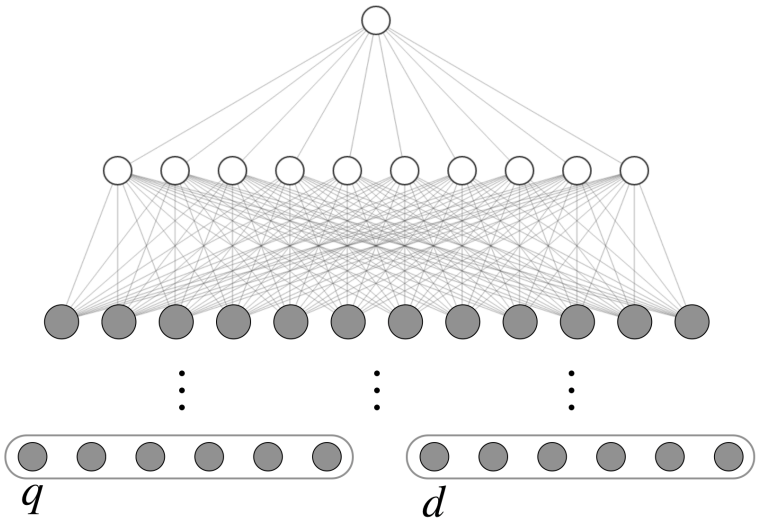

Figure 2.4: For the purposes of computing training point influence, the neural network is treated as a logistic regression on the bottleneck features (the input to the last layer of the network). Grey circles indicate frozen layers.

As in [Koh and Liang, 2017], s_{test}, defined in equation 2.10 is computed for every z_{test} using the method of conjugate gradients following [Shewchuk, 1994] using the magnitude of the residual as the stopping condition. A small damping term is added to the diagonal of the Hessian to ensure that it is positive definite, otherwise, negative curvature in the objective function causes Conjugate Gradient (CG) to diverge [Martens, 2010]. Experiments using preconditioning of the Hessian with a diagonal matrix whose elements are those of the Laplacian did not show significantly improved computational stability. Following [Martens, 2010], the HVP operations required in computing s_{test} are sped up by initializing the conjugate gradient algorithm with the solution from the previous computation.

Also note that the size of the dataset makes computing the HVP impractical, accordingly, a subset of the training data is used as a mini-batch to compute an estimate of the Hessian. As recommended by [Martens, 2010], this estimate is

kept constant across iterations of the CG algorithm to improve stability. The full algorithm is presented in algorithm 2.

Algorithm 2: Training the *influence-aware* model

Input: Set of training points $\mathcal{Z}_{train}$ and test points $\mathcal{Z}_{test}$
Output: The trained *influence-aware* model

Train the model on the full training set, $\mathcal{Z}_{train}$
for $z_{test_i} \in \mathcal{Z}_{test}$ **do**
  $s_{test_i} \leftarrow \mathrm{CG}(H_{\hat{\theta}}, \nabla_\theta L(z_{test_i}; \hat{\theta}))$
end
$\mathcal{Z}_{clean} \leftarrow \{\}$
for $z_{train_i} \in \mathcal{Z}_{train}$ **do**
  for $z_{test_i} \in \mathcal{Z}_{test}$ **do**
    $\mathcal{I}_{drop}(z_{test_i}, z_{train_i}) \leftarrow s_{test_i}^T \nabla_\theta L(z_{train}; \hat{\theta})$
  end
  $\mathcal{I}_{drop}(z_{train_i}) \leftarrow \sum_{z_{test_i} \in \mathcal{Z}_{test}} \mathcal{I}_{drop}(z_{test_i}, z_{train_i})$
  if $\mathcal{I}_{drop}(z_{train_i}) < 0$ **then**
    $\mathcal{Z}_{clean} \leftarrow \mathcal{Z}_{clean} \cup \{z_{train_i}\}$
  end
end
Retrain the model on the clean training set, $\mathcal{Z}_{clean}$

Chapter 3

Experimental Evaluation

The proposed methods give alternative training strategies for the *rank* model of [Dehghani et al., 2017c] giving us the *noise-aware* and *influence-aware* models. Evaluation is performed on the standard Robust04 corpus with the associated test queries and relevance labels. The Robust04 dataset contains 528,000 primarily news related documents whose average length is 254 tokens. The dataset also contains 250 queries with corresponding relevance labels.

3.1 Setup

As in [Dehghani et al., 2017c], the training queries for the models evaulated in this work comes from the AOL query logs [Pass et al., 2006] which is a collection of queries performed by real users over a 3 month period in 2006 on the AOL search website of the time. All queries containing URLs and URL substrings such as "http://" and ".com" are omitted. Instead of learning a vector representation for each unique word that appears in the corpus and set of queries, tokens that appear less than 10 times in the corpus are mapped to a special "UNK" token to represent

that it is OoV. All training queries that contain "UNK" tokens are discarded. After this filtering, the training set contains $\sim 3 \times 10^6$ queries.

The popular Indri search engine is used to conduct indexing and retrieval for all experiments. The QL retrieval model using the default parameters [Ponte and Croft, 1998] is used as the weak supervision source for the *rank* and *influence-aware* models (see 3.2 for details regarding the weak supervision sources used for the *noise-aware* model). The top 10 relevant documents from each ranking are retrieved via Indri for the desired retrieval method. As a comparison, previous works trained on as many as the top 1000 documents for each query. To compensate for this difference, n_{neg} additional documents are randomly sampled uniformly from the rest of the corpus for each of these 10 documents. The model under test is trained on a random subset of 100k rankings generated by this process. This is fewer than 10% the number of rankings used in previous works [Nie et al., 2018, Dehghani et al., 2017c], each of which also contains far fewer document pairs.

The `840B.300d` GloVe [Pennington et al., 2015] pretrained word embedding set[1] is used for the word embedding representations, W. Experiments using Word2Vec [Mikolov et al., 2013] gave similar behavior but performed worse overall on the target metrics. The feed-forward network hidden layer sizes are chosen from {512, 256, 128, 64} with a maximum of 5 layers. The first 50 queries of the Robust04 dataset (topics 301-350) as the development set for hyperparameter selection, computation of $\mathcal{I}_{drop}$ and early stopping. The remaining 200 queries are used for evaluation.

Early stopping is performed by monitoring the loss at the development set at the end of each epoch. When the development set loss increases between epochs, training is stopped and the model from the end of the last epoch is used.

[1] https://nlp.stanford.edu/projects/glove/

For the pointwise setup, the document scores are normalized per query such that scores are on the same scale across different queries. This improves the performance of the pointwise setup slightly since the exact score is irrelevant as long as the rank is preserved. Coercing document scores into the same range across queries is an easier regression problem and reduces the risk of overfitting as well as the impact of outliers. More complex techniques for normalizing the document scores, such as monotone retargeting [Acharyya et al., 2012], could also be used. In the experimental results that follow, normalization is performed by taking the softmax over the query likelihood scores:

$$score_{normalized\ QL}(d, q) = \frac{\exp(score_{QL}(d, q))}{\sum_{d' \in \tau_q} \exp(score_{QL}(d', q))}$$

where τ_q is the set of results retrieved by the Indri retrieval model.

During inference, documents are ranked by the output of the feed-forward network. Since it is not feasible to rank all the documents in the corpus, the top 100 documents are fetched using the QL retrieval model and then reranked using the trained model's scores. It is typical for LeToR models to approach ad-hoc retrieval in this reranking format [Mitra and N Craswell, 2018].

3.2 Model Specific Details

For the *noise-aware* model, separate rankings are generated for each query using the following retrieval methods: Okapi BM25, TF-IDF, QL, QL+RM3 [Abdul-Jaleel et al., 2004] using Indri with the default parameters. The weights in (2.7) are learned using the rankings from the first 10k queries. During training, the soft labels are computed from Λ and the learned parameters w as described in section 2.2.

For the *influence-aware* model, the *rank* model is first trained on the full dataset. Then $\mathcal{I}_{drop}(x_i)$ is computed for each training point. A new "cleaned" dataset is formed by dropping all training examples with a negative value for $\mathcal{I}_{drop}(x_i)$. This turns out to typically be around half of the original training set. The model is then retrained on this subset, giving the *influence-aware* model.

The *rank* model is trained to minimize the hinge loss which requires specification of the margin ϵ. Interestingly, using a smaller margin, ϵ, in the training loss of the *rank* model leads to improved performance. The choice of a smaller margin incurs 0 loss for a smaller difference in the model's relative preference between the two documents. Intuitively, this allows for less overfitting to the noisy data. A margin of 0.1 is chosen by cross-validation.

The *noise-aware* and *influence-aware* models train end-to-end in around 12 and 15 hours respectively on a single NVIDIA Titan Xp.

3.3 Results

The two methods introduced in this work are compared against two baselines, the unsupervised ranker (QL) and the *rank* model. Compared to the other unsupervised rankers (see section 3.2) used as input to the *noise-aware* model, the QL ranker performs the best across all metrics (NDCG, MAP and Precision@10). The ranking methods are compared before and after linear interpolation smoothing with the normalized QL document scores.

The results in tables 3.1 and 3.2 show that the *noise-aware* and *influence-aware* models perform similarly, with both outperforming the unsupervised baseline. Figure 3.1 shows that the *rank* model quickly starts to overfit. This does not contradict the results in [Dehghani et al., 2017c] since the setup here trains the models on

Table 3.1: Results comparison with smoothing. Bold items are the largest in their row and daggers indicate statistically significant improvements over the *rank* model at a level of 0.05 using Bonferroni correction.

Metric	Rank Model	Noise-Aware	Influence-Aware	QL	Pointwise-Rank Model
NDCG@10	0.3881	† 0.3952	†**0.4008**	0.3843	0.3863
Prec@10	0.3535	† 0.3621	†**0.3657**	0.3515	0.3505
MAP	0.2675	† 0.2774	†**0.2792**	0.2676	0.2675

Table 3.2: Results comparison without smoothing. Bold items are the largest in their row and daggers indicate statistically significant improvements over the *rank* model at a level of 0.05 using Bonferroni correction.

Metric	Rank Model	Noise-Aware	Influence-Aware	Pointwise-Rank Model
NDCG@10	0.2610	† 0.2886	†**0.2966**	0.1988
Prec@10	0.2399	†**0.2773**	† 0.2742	0.1995
MAP	0.1566	† 0.1831	†**0.1839**	0.1025

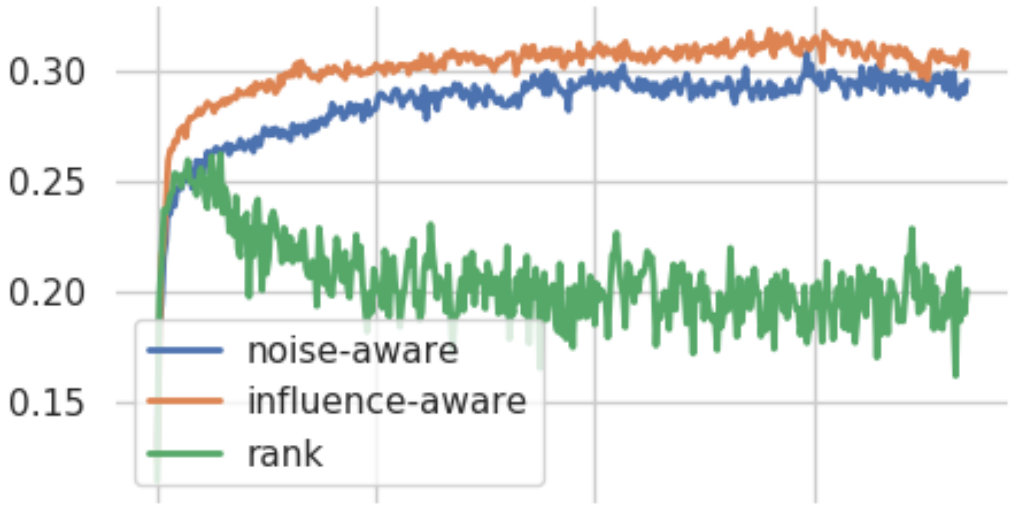

Figure 3.1: Test NDCG@10 during training

far fewer pairs of documents for each query, so each relevance label error has much greater impact. For each query, the distribution over documents is uniform outside the results from the weak supervision source, so worse performance is expected than if using a more faithful relevance distribution. The proposed approaches use an improved estimate of the relevance distribution at the most important positions in the ranking, allowing them to perform well.

3.3.1 Representative Examples

In order to gain a more concrete understanding of how the *noise-aware* and *influence-aware* methods are helping improve the trained model's performance, following are two representative training examples showing how each of the proposed methods overcomes the limitations of the *rank* model.

Example 3.3.1. The method in section 2.2 used to create labels for the *noise-aware* model gives the following training example an unconfident label (~ 0.5) rather than a relevance label of 1 or 0: (q="town of davie post office", (d_1=`FBIS3-25584`, d_2=`FT933-13328`)) where d_1 is ranked above d_2. Both of these documents are about people named "Davie" rather than about a town or a post office, so it is reasonable to avoid specifying a hard label indicating which one is explicitly more relevant.

Example 3.3.2. One of the most harmful training points as determined by the method described in section 2.3 (used to drop training examples for the *influence-aware* model) is the pair (q="pictures of easter mice", (d_1=`FT932-15650`, d_2=`LA041590-0059`)) where d_1 is ranked above d_2. d_1 discusses the computer input device and d_2 is about pictures that are reminiscent of the holiday. The incorrect relevance label explains why the method identifies this as a harmful training example.

Table 3.3: Results comparison with and without smoothing after fine-tuning.

Metric	Rank Model with smoothing	Rank Model without smoothing
NDCG@10	0.3851	0.1972
Prec@10	0.3520	0.1929
MAP	0.2650	0.1045

Table 3.4: Comparison of scores without smoothing when flipping 50% of labels in the training set. Bold items are the largest in their row and daggers indicate statistically significant improvements over the *rank* model at a level of 0.05 using Bonferroni correction.

Metric	Rank Model	Noise-Aware	Influence-Aware
NDCG@10	0.2000	† **0.2828**	† 0.2795
Prec@10	0.2010	† **0.2626**	† 0.2540
MAP	0.1058	† **0.1742**	† 0.1715

3.3.2 Fine-Tuning

In training of the *rank* model, the 50 ranking with gold relevance labels are used only for hyperparameter selection and early stopping. A better approach might be to instead use these relevance labels to fine tune the model which is pretrained on the weakly supervised training set. Since these gold labels do not provide full rankings, only 0 or 1 relevance labels, the fine-tuning training set is created by taking pairs of documents, (d_1, d_2) where the relevance label of d_1 and d_2 are 1 and 0 respectively. For each of these pairs $rel_{q,(d_i,d_j)} = 1$. This gives us approximately 4k training examples for fine tuning. The results are presented in table 3.3 for the *rank* model. Despite use of fine-tuning techniques such as cyclical learning rates [Howard and Ruder, 2018], the performance degraded after training on this small set of labels.

3.3.3 Handling Intentionally Mislabeled Examples

To further isolate the ability of the *noise-aware* and *influence-aware* models to detect and handle mislabeled training examples, this section investigates the impact of manually flipping half of the relevance labels in the weakly supervised training set.

For the *noise-aware* model, each nonzero value in the Λ matrix formed from the unsupervised retrieval models is flipped to the opposite sign with probability $\frac{1}{2}$. The results on the test set are shown in table 3.4. Both the *influence-aware* model and the *noise-aware* model are able to maintain acceptable performance while the performance of the *rank* model seriously degrades. In this case the *noise-aware* model outperforms the *influence-aware* model. Intuitively, this makes sense since the noise applied to each of the weak supervision sources of the *noise-aware* model is i.i.d. and thus even simple majority voting will converge to the true label as the number of weak supervision sources increases.

Chapter 4

Discussion and Future Work

The key limitation to previous work that leverages weak supervision sources for ad-hoc retrieval is the size of the training set used. Although the dataset can be generated at relatively low cost, training a neural network on this dataset is arguably not. Rather than resort to costly distributed computing to learn the parameters of the ranking model, another option is to make more effective use of the computational resources at hand.

This work presents two approaches that reduce the computational burden of learning a ranking model for ad-hoc retrieval using weak training data. In both cases, this is achieved by reducing the amount of training data required to surpass the performance of the original unsupervised method that generated the training data. The models presented are trained on 10% of the training queries as compared to the baseline model from prior work. Furthermore, each of the training rankings used in this work are themselves one-tenth the length of the rankings used in previously developed techniques. Note that reducing the length of the training rankings leads to a more significant speed-up in training time than reducing the number of training

queries since the former scales as $\mathcal{O}(n^2)$ while the latter scales as $\mathcal{O}(n)$.

Intuitively, each of the methods introduced here trains the model on an improved estimate of the relevance distribution of each training query. The soft labels of the *noise-aware* approach incorporate a notion of uncertainty into the relevance scores while the *influence-aware* approach simply skips training examples deemed harmful to generalization performance.

Comparing the two proposed methods, the *noise-aware* model does not require ground truth labels, but has an additional data dependency on multiple unsupervised rankers while the *influence-aware* model requires a small set of human-curated rankings in addition to a re-train of the model, although experimentally, only around half the dataset is used when training the second time around.

Interesting paths for future work involve learning a better joint distribution for training the *noise-aware* model or leveraging ideas from [Zamani et al., 2018] to construct soft labels. A natural extension is to use a more complex model of the joint distribution, perhaps incorporating information from the feature space into 2.8. Similarly, we could apply ideas from unsupervised LeToR [Bhowmik and Ghosh, 2017] to form better noise-aware labels by taking advantage of item attributes. Another interesting direction of work is to model the position of documents in the ranked list during the learning of the joint distribution from 2.2 in order to take advantage of the contextual information available in a ranking. A Cranking [Lebanon and Lafferty, 2002] type approach could be taken to model the labelers as drawing permutations from a Mallows model rather than Bernoulli variables.

For the *influence-aware* model, the softrank loss [Baeza-Yates et al., 2007], rather than cross-entropy, could be used to obtain a better approximation of the impact of a single training example on the final model's performance. Similar to

approaches suggested above for the *noise-aware* model, this would more accurately frame the problem as a ranking task, rather than as a binary classification task. In addition, rather than computing the influence of a single training example, harmfulness could instead be assessed by computing the influence of a set of training points on the collective set of test points [Khanna et al., 2018]. This would give a more faithful approximation to the retrained model's performance since it considers the impact of dropping a set of training points as a whole, rather than individually.

Little effort was put into selecting the set of queries used to generate the training rankings. Intuitively, some queries will be more helpful than others in training a high quality ranking model. For example, the query 'mozilla firefox' from the training set likely does not have very many relevant documents in the corpus so the ranking given by the QL retrieval model will not be very informative for the model being trained. [Zamani et al., 2018] tackles the query performance prediction task using a neural network and achieve state of the art results. One could prioritize training examples of the *rank* model by the predicted performance and potentially achieve a speed-up in training time.

Bibliography

[Abdul-Jaleel et al., 2004] Abdul-Jaleel, N., Allan, J., Croft, B., Diaz, F., Larkey, L., Li, X., Metzler, D., Smucker, M. D., Strohman, T., Turtle, H., and Wade, C. (2004). Umass at trec 2004: Notebook. *academia.edu*.

[Acharyya et al., 2012] Acharyya, S., Koyejo, O., and Ghosh, J. (2012). Learning to rank with bregman divergences and monotone retargeting. *arXiv preprint arXiv:1210.4851*.

[Ai et al., 2018] Ai, Q., Bi, K., Luo, C., Guo, J., and Croft, W. B. (2018). Unbiased Learning to Rank with Unbiased Propensity Estimation. In *The 41st International ACM SIGIR Conference*, pages 385–394, New York, New York, USA. ACM Press.

[Baeza-Yates et al., 2007] Baeza-Yates, R., Ribeiro, B. d. A. N., et al. (2007). Learning to rank with nonsmooth cost functions. *NIPS*.

[Berend and Kontorovich, 2013] Berend, D. and Kontorovich, A. (2013). Consistency of weighted majority votes. *arXiv.org*.

[Bhowmik and Ghosh, 2017] Bhowmik, A. and Ghosh, J. (2017). LETOR Methods for Unsupervised Rank Aggregation. In *the 26th International Conference*, pages 1331–1340, New York, New York, USA. ACM Press.

[Blei et al., 2002] Blei, D. M., Ng, A. Y., and information, M. J. (2002). Latent dirichlet allocation. *papers.nips.cc*.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). Convex optimization.

[Burges, 2010] Burges, C. (2010). From Ranknet to Lambdarank to Lambdamart: An Overview.

[Cao et al., 2006] Cao, Y., Xu, J., Liu, T.-Y., Li, H., Huang, Y., and Hon, H.-W. (2006). Adapting ranking SVM to document retrieval. In *the 29th annual international ACM SIGIR conference*, pages 186–193, New York, New York, USA. ACM.

[Cook, 1986] Cook, R. D. (1986). Assessment of Local Influence. *Journal of the Royal Statistical Society: Series B (Methodological)*, 48(2):133–155.

[Cook and Weisberg, 1982] Cook, R. D. and Weisberg, S. (1982). *Residuals and influence in regression*. New York: Chapman and Hall.

[Croft et al., 2010] Croft, W. B., Metzler, D., and Strohman, T. (2010). Search engines: Information retrieval in practice.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

[Dehghani et al., 2017a] Dehghani, M., Mehrjou, A., Gouws, S., Kamps, J., and Schölkopf, B. (2017a). Fidelity-Weighted Learning. *arXiv.org*.

[Dehghani et al., 2017b] Dehghani, M., Severyn, A., Rothe, S., and Kamps, J. (2017b). Learning to Learn from Weak Supervision by Full Supervision. *arXiv.org*, pages 1–8.

[Dehghani et al., 2017c] Dehghani, M., Zamani, H., Severyn, A., Kamps, J., and Croft, W. B. (2017c). Neural Ranking Models with Weak Supervision. In *the 40th International ACM SIGIR Conference*, pages 65–74, New York, New York, USA. ACM Press.

[Harris, 2015] Harris, Z. S. (2015). Distributional Structure. *WORD*, 10(2-3):146–162.

[Howard and Ruder, 2018] Howard, J. and Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *arXiv.org*.

[Huang and Gao, 2013] Huang, P.-S. and Gao, J. (2013). Learning deep structured semantic models for web search using clickthrough data. ACM International Conference on Information and Knowledge Management (CIKM).

[Jiang et al., 2017] Jiang, X., Pan, S., Long, G., Xiong, F., Jiang, J., and Zhang, C. (2017). Cost-sensitive learning with noisy labels. *JMLR*.

[Joachims, 2002] Joachims, T. (2002). Optimizing search engines using clickthrough data. New York, New York, USA. ACM.

[Khanna et al., 2018] Khanna, R., Kim, B., Ghosh, J., and Koyejo, O. (2018). Interpreting Black Box Predictions using Fisher Kernels. *arXiv.org*.

[Koh and Liang, 2017] Koh, P. W. and Liang, P. (2017). Understanding Black-box Predictions via Influence Functions. *arXiv.org*, pages 1–11.

[Lavrenko et al., 2001] Lavrenko, V., Forum, W. C. A. S., and 2017 (2001). Relevance-based language models. *dl.acm.org*.

[Le and Mikolov, 2014] Le, Q. V. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. *arXiv.org*.

[Lebanon and Lafferty, 2002] Lebanon, G. and Lafferty, J. (2002). Cranking: Combining rankings using conditional probability models on permutations. *Citeseer*.

[Liu, 2009] Liu, T.-Y. (2009). Learning to Rank for Information Retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331.

[Lv and Zhai, 2009] Lv, Y. and Zhai, C. (2009). A comparative study of methods for estimating query language models with pseudo feedback. *Proceeding of the 18th ACM conference*, pages 1895–1898.

[Martens, 2010] Martens, J. (2010). Deep learning via Hessian-free optimization.

[Metzler et al., 2004] Metzler, D., Lavrenko, V., SIGIR, W. C., and 2004 (2004). Formal multiple-bernoulli models for language modeling. *ciir.cs.umass.edu*.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. pages 3111–3119.

[Mitra and N Craswell, 2018] Mitra, B. and N Craswell (2018). An introduction to neural information retrieval. *nowpublishers.com*.

[Nie et al., 2018] Nie, Y., Sordoni, A., and Nie, J.-Y. (2018). Multi-level Abstraction Convolutional Model with Weak Supervision for Information Retrieval. In *The 41st International ACM SIGIR Conference*, pages 985–988, New York, New York, USA. ACM Press.

[Northcutt et al., 2017] Northcutt, C. G., Wu, T., and Chuang, I. L. (2017). Learning with Confident Examples: Rank Pruning for Robust Classification with Noisy Labels. *arXiv.org*.

[Pass et al., 2006] Pass, G., Chowdhury, A., and Torgeson, C. (2006). A Picture of Search. *Infoscale*, pages 1–es.

[Pearlmutter, 1994] Pearlmutter, B. (1994). Fast exact multiplication by the Hessian. *MIT Press*, 6(1):147–160.

[Pennington et al., 2015] Pennington, J., Socher, R., and Manning, C. D. (2015). GloVe: Global Vectors for Word Representation.

[Ponte and Croft, 1998] Ponte, J. M. and Croft, W. B. (1998). A Language Modeling Approach to Information Retrieval. *SIGIR*, pages 275–281.

[Ratner et al., 2017] Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. (2017). Snorkel. *Proceedings of the VLDB Endowment*, 11(3):269–282.

[Robertson et al., 1995] Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., Gatford, M., et al. (1995). Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

[Shewchuk, 1994] Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain.

[Taylor et al., 2008] Taylor, M., Guiver, J., Robertson, S., 2008, T. M. P. o. t., and 2008 (2008). Softrank: optimizing non-smooth rank metrics. *dl.acm.org*.

[Zamani and Croft, 2017] Zamani, H. and Croft, W. B. (2017). Relevance-based Word Embedding. In *the 40th International ACM SIGIR Conference*, pages 505–514, New York, New York, USA. ACM Press.

[Zamani and Croft, 2018] Zamani, H. and Croft, W. B. (2018). *On the Theory of Weak Supervision for Information Retrieval*. ACM, New York, New York, USA.

[Zamani et al., 2018] Zamani, H., Croft, W. B., and Culpepper, J. S. (2018). Neural Query Performance Prediction using Weak Supervision from Multiple Signals. In *The 41st International ACM SIGIR Conference*, pages 105–114, New York, New York, USA. ACM Press.